◊ .1

Halftoning with Selective Precipitation and Adaptive Clustering

Tien-tsin Wong
Computer Science Department
The Chinese University of Hong Kong
Shatin, Hong Kong
ttwong@cs.cuhk.hk

Siu-chi Hsu
Creature House, Ltd.
Hong Kong
schsu@acm.org

Halftoning techniques are used to display continuous tone pictures on bi-level displays and printers (or on those with very limited number of shades). The most popular and well-known techniques are ordered dither and error diffusion. The latter produces aperiodic patterns with limited low frequency components, a useful property (Ulichney 1987), but its dispersed dots suffer from an excessive smudging, which is especially objectionable on high resolution devices. Ordered dither, on the other hand, is capable of clustering the dots produced by using a properly designed dither matrix. However, a regular dither pattern is then clearly visible in the output picture. A comparison of most digital halftoning techniques can be found in the literature (Schumacher 1991) (Ulichney 1987).

Recently, researchers have been investigating new halftoning techniques which traverse images along a space filling curve (Witten and Neal 1982) (Cole 1990) (Wyvill and McNaughton 1991) (Velho and de Miranda Gomes 1991) (Zhang and Webber 1993). The space filling curve halftoning is attractive because of its pleasant smooth grains in the resultant image and the aperiodicity of the halftone pattern. Velho and de Miranda Gomes (*op. cit.*) further proposed a clustered-dot space filling curve halftoning algorithm which reduces the smudging problem. However, clustering the dots naïvely would blur the image excessively. This gem presents two improvements, *selective precipitation* and *adaptive clustering*, to minimize the blurring phenomenon.

◊ Selective Precipitation ◊

The first improvement is to precipitate black dots selectively. The original clustered-dot space filling curve halftoning algorithm precipitates the black dots at a fixed location, say, at the beginning of each cluster. This results in a poor approximation to the original

ISBN 0-12-XXXXXX-X

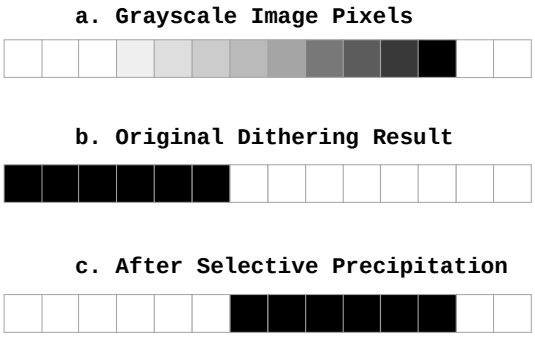

Figure 1. Halftoning (a 1-D continuous tone image) using precipitation.

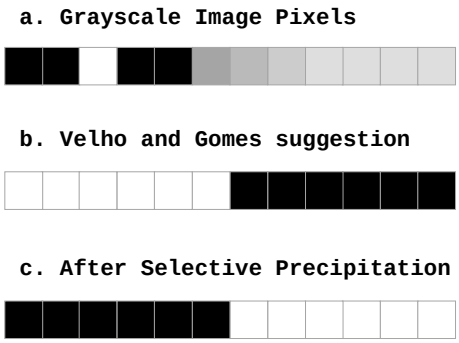

Figure 2. Halftoning using selective precipitation.

image when the original gray values in a particular cluster are not gathered around that fixed location (Figure 1). Although Velho and de Miranda Gomes have briefly suggested that the white subregion can be centred at the pixel with the highest intensity in order to preserve details, this may still result in a poor approximation (Figure 2).

By placing the black output dots over the area with the highest *total* gray value, a better approximation can be obtained. This technique is called *selective precipitation*. The number of black dots to be output in the current cluster is determined by summing all gray values inside the cluster. This number is then used as the length of a moving window which shifts within the halftone cluster. The objective is to find the position of the moving window having the highest summed gray pixel value. The black dots are then precipitated at that position.

In essence, spatial offsets are applied to localize the position of maximum dot density. This approach advances the original ARIES technique (Pryor et al. 1978) researched extensively at Xerox (Roetling 1976). The basic algorithm is sketched below.

Input

1. `input[]`: a one-dimensional array of continuous tone pixels on the range $[0 \ldots 1]$ presented as a one-dimensional array in the order of the space filling traverse.
2. `clustersize`: the cluster size
3. `clusterstart`: the index of the current cluster's first element.
4. `graysum`: cumulative gray sum within the current cluster.

```
winlen := ⌊graysum⌋
graysum := graysum - winlen
winsum := 0
maxsum := 0
winstart := clusterstart
for i := winstart to (winstart+winlen-1) do
begin
     winsum := winsum + input[i]
end
while (winstart+winlen) - clusterstart < clustersize
begin
     if maxsum < winsum
     begin
          maxsum := winsum
          rightplace := winstart
     end
     winsum := winsum - input[winstart] + input[winstart+winlen]
     winstart := winstart + 1
end
```

Output

1. Black dots are produced at `rightplace` for `winlen` positions.
2. The final quantization error in `graysum`.

The time complexity of this process is clearly linear.

♢ Adaptive Clustering ♢

Another factor which causes the blurring is the rigid grouping of output black dots (Figure 3). Here, the original gray values are grouped at opposite ends of the cluster. Presented such data, selective precipitation can generate black dots only at the one end having a higher total gray value. A better approximation can be obtained by dividing the cluster into two smaller clusters and performing the selective precipitation process in both clusters.

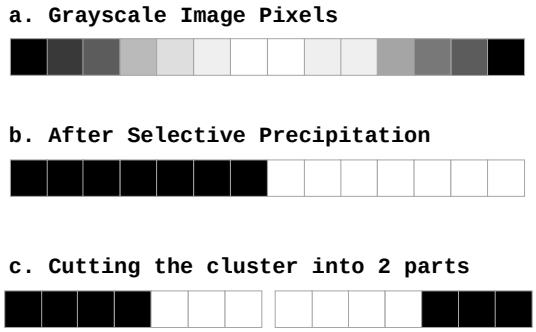

Figure 3. Selective precipitation with adaptive clustering.

One method of locating the point of subdivision is finding the sharp edges. Since human eyes are more sensitive to high frequency changes, blurring phenomena on sharp edges are more noticeable. A partitioning of clusters at sharp edges therefore preserves sharp details. This approach is used; the improvement is called *adaptive clustering*.

Since the space filling curve goes through each pixel in the image exactly once, it effectively scales down the 2-D edge detection problem into a 1-D problem. It is therefore sufficient to employ merely a 1-D filter along the space filling curve in order to detect sharp edges. That is, the curve's traverse constitutes a continuous image signal. Applying the standard 1-D negative of the Laplacian of the Gaussian filter (Jain 1989) can detect these sharp edges along the chain (signal). The formula of the filter is

$$\frac{\exp(-x^2/2\sigma^2)}{\sigma^3\sqrt{2\pi}}\left(1-\frac{x^2}{\sigma^2}\right),$$

where σ is the standard deviation and x is the input signal. A filter kernel with a width of seven pixels ($\sigma = 1$) is sufficient.

The adaptive clustering algorithm is now outlined. Traverse the image pixels along a chosen space filling cover, forming a cluster whenever `N` (the maximum cluster size) pixels have been traversed or a sharp edge is encountered, whichever comes first. Perform selective precipitation upon the current cluster. The pseudocode follows.

Input

1. `N`: maximum cluster size
2. `T`: threshold
3. `M`: number of input pixels
4. `input[1..M]`: 1-D pixel data in preselected order

```
graysum := 0
clustersize := 0
clusterindex := 0
lastconvol := 0
for index := 0 to M-1 do
begin
     convol := InvLaplGaussian(input, 7, index-3)
                                        Convolve array with seven sample window
                                        centered about current pixel.
     graysum := graysum + input[index]  Accumulate total gray.
     clustersize := clustersize + 1     Increase current cluster.
     if |convol-lastconvol| > T or clustersize > N
     begin
          precipitate(input, graysum, clustersize, clusterindex);
                                        Perform selective precipitation outlined
                                        in the previous pseudocode.
          clustersize := 0              Begin next cluster.
          clusterindex := index
     end
     lastconvol := convol
end
```

The sensitivity of the edge detection filter affects the resulting halftone image and may be controlled with a user-defined threshold `T`. This value can also be determined automatically using previous techniques (Schlag 1991). A lower threshold detects additional edges, resulting in potentially smaller clusters.

Figures 4 and 5 show the performance of the improved halftoning method. Note the excessive blurring, seen as a loss of floor texture (Figure 4(b)) or of fine image detail (Figure 5(b)). This blurring phenomenon is significantly reduced when selective precipitation and adaptive clustering is employed (Figure 4(c) and 5(c), respectively).

(a)

(b)

(c)

Figure 4. Teapot. (a) Original gray scale image (256x256). (b) Space filling dithering; cluster size N = 9 pixels. (c) Selective precipitation with adaptive clustering; N = 9.

(a)

(b)

(c)

Figure 5. F16 factory. (a) Original gray scale image (256x256). (b) Space filling dithering; cluster size N = 9 pixels. (c) Selective precipitation with adaptive clustering; N = 9.

♢ C Implementation ♢

```
/*=========================================================================*
 * Halftoning using Space Filling Curve with adaptive clustering and       *
 * selective precipitation                                                 *
 *                                                                         *
 * Limitation:                                                             *
 * Only process image with size 2^n x 2^n where n is positive integer.     *
 *=========================================================================*/

unsigned char **path;    /* space filling curve path */
/*
 * path[] is a global array storing the information to move along
 *        the space filling curve.
 * genspacefill() is a function to generate the information in path[].
 *        This function is implemented based on a gem in Graphics Gems II,
 *        Ken Musgrave, "A Peano Curve Generation Algorithm".
 * move() is a macro to move along the space filling curve using the
 *        the information stored in path[].
 */
#define TRUE    1
#define FALSE   0
#define BLACK   255
#define WHITE   0
#define LEFT    0
#define RIGHT   1
#define UP      2
#define DOWN    3
#define END     255
#define move(x,y)  switch (path[x][y])          \
                   {                            \
                     case UP:   y++; break;     \
                     case DOWN: y--; break;     \
                     case LEFT: x--; break;     \
                     case RIGHT:x++; break;     \
                   }

/*
 * Description of parameters:
 *   picture,         2D array holding the grayscale image.
 *   out,             2D array holding the dithered image.
 *   maxclustersize,  Max cluster size, N.
 *   thresh,          Edge detection threshold T.
 *   do_sp,           Flag to switch on/off selective precipitation.
 *                    To switch off the selective precipitation,
 *                    set do_sp = FALSE.
 *   do_ac,           Flag to switch on/off adaptive clustering.
 *                    To switch off the adaptive clustering, set do_ac=FALSE
 */
void spacefilterwindow(int **picture, int **out, int maxclustersize,
                       int thresh, char do_sp, char do_ac)
{
  char edge;             /* Flag indicate sudden change detected */
```

```
char ending;           /* flag indicates end of space filling curve */
int accumulator;       /* Accumulate gray value */
int currclustersize;   /* Record size of current cluster */
int frontx, fronty;    /* Pointer to the front of the cluster */
int windowx, windowy;  /* Pointer to first pixel applied with filter */
int clusterx, clustery;/* Pointer to first pixel in current cluster */
int windowlen;         /* Size of the moving window */
int winsum;            /* Current moving window's sum */
int maxsum;            /* Maximum moving window's sum recorded */
int rightplace;        /* Position of the moving window with max sum */
int *cluster;          /* An array hold the pixel of current cluster */
int last, i,j, tempx, tempy, currx, curry;    /* temp variables */
long filter[7] = {-1, -5, 0, 13, 0, -5, -1};  /* 1D -ve Lap. Gauss. filter */
long convolution;      /* Convolution value in this turn */
long lastconvolution;  /* Convolution value in last turn */
/*
 * Description of the pointer along the space filling curve.
 *
 * clusterx,                          windowx,    currx,     frontx,
 * clustery                           windowy     curry      fronty
 *    |                                  |          |          |
 *    v                                  v          v          v
 *   +-----------------------------------------------------------+
 *   |                       Cluster                             |
 *   +-----------------------------------------------------------+
 *                                      |                       |
 *                                      |                       |
 *                                      |          /\           |
 *                                      |        /    \         |
 *                                      |___   /        \   ___|
 *                                      |    \/          \/   |
 *                                   -ve Laplacian of Gaussian Filter
 */

if ((cluster=malloc(sizeof(int)*maxclustersize))==NULL)
{
  fprintf(stderr,"not enough memory for cluster\n");
  return;
}
genspacefill();    /* generates the spacefilling path */

convolution=0;
currclustersize=0;
accumulator=0;
for (frontx=0, fronty=0, i=0 ; i<7 ; i++)
{
  if (i<3)
  {
    cluster[currclustersize] = picture[frontx][fronty];
    accumulator += cluster[currclustersize];
    currclustersize++;
  }
  if (i==3)
```

```
    {  currx = frontx;  curry = fronty;  }
    convolution += filter[i]*(long)(picture[frontx][fronty]);
    move(frontx,fronty);  /* assume the image at least has 7 pixels */
  }
  lastconvolution = convolution;
  clusterx=0;   clustery=0;
  windowx=0;    windowy=0;
  edge=FALSE;
  ending=FALSE;

  while (TRUE)
  {
    if (do_ac) /* switch on/off adaptive clustering */
    {
      /* do convolution */
      convolution = 0;
      for (tempx=windowx, tempy=windowy, i=0 ; i<7 ; i++)
      {
        convolution += filter[i]*picture[tempx][tempy];
        move(tempx,tempy);
      }

      /* detect sudden change */
      if ( (convolution >= 0 && lastconvolution <=0
            && abs(convolution-lastconvolution)>thresh)
         ||(convolution <= 0  && lastconvolution >=0
            && abs(convolution-lastconvolution)>thresh))
        edge=TRUE; /* force output dots */
    }

    /* Output dots if necessary */
    if (edge || currclustersize >= maxclustersize || ending)
    {
      edge=FALSE;

      /* Search the best position within cluster to precipitate */
      rightplace = 0;
      if (do_sp) /* switch on/off selective precipitation */
      {
        windowlen = accumulator/BLACK;
        winsum = 0;
        for (i=0; i<windowlen; i++)
          winsum += cluster[i];
        for (maxsum=winsum, last=0; i<currclustersize; i++, last++)
        {
          winsum+= cluster[i] - cluster[last];
          if (winsum > maxsum)
          {
            rightplace=last+1;
            maxsum=winsum;
          }
        }
      }
```

```
      /* Output dots */
      for (i=0 ; currclustersize!=0 ; currclustersize--, i++)
      {
        if (accumulator>=BLACK && i>=rightplace)  /* precipitates */
        {
          out[clusterx][clustery]=BLACK;
          accumulator-=BLACK;
        }
        else
          out[clusterx][clustery]=WHITE;
        move(clusterx,clustery)
      } /* for */

      if (ending)
        break;
    } /* if */

    cluster[currclustersize] = picture[currx][curry];
    accumulator += cluster[currclustersize];
    currclustersize++;
    if (path[currx][curry]==END)
      ending = TRUE;
    move(currx,curry);
    move(windowx,windowy);
    move(frontx,fronty);
  } /* while */
}
```

♢ Bibliography ♢

(Cole 1990) A. J. Cole. Naïve halftoning. In T. S. Chua and Kunii, editors, *Proceedings of CG International '90*, pages 203–222. Springer-Verlag, 1990.

(Jain 1989) Anil K. Jain. *Fundamentals of Digital Image Processing.* Prentice Hall, 1989.

(Pryor et al. 1978) R. W. Pryor, G. M. Cinque, and A. Rubenstein. Bi-level displays: A new approach. *Proc. Soc. Info. Disp. (SID)*, 19:127–131, 1978.

(Roetling 1976) Paul J. Roetling. Halftone method with edge enhancement and moiré suppression. *Jour. Opt. Sco. Amer.*, 66(10):985–989, October 1976.

(Schlag 1991) John Schlag. Noise thresholding in edge images. In James Arvo, editor, *Graphics Gems II*, page 105. Academic Press, 1991.

(Schumacher 1991) Dale A. Schumacher. A comparsion of digial halftoning techniques. In James Arvo, editor, *Graphics Gems II*, pages 57–77. Academic Press, 1991.

(Ulichney 1987) R. Ulichney. *Digital Halftoning*. MIT Press, 1987.

(Velho and de Miranda Gomes 1991) Luiz Velho and Jonas de Miranda Gomes. Digital halftoning with space filling curves. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 81–90, July 1991.

(Witten and Neal 1982) I. H. Witten and R. M. Neal. Using Peano curves for bilevel display of continuous-tone images. *IEEE Computer Graphics and Applications*, 2:47–52, may 1982.

(Wyvill and McNaughton 1991) Geoff Wyvill and Craig McNaughton. Three plus five makes eight: A simplified approach to halftoning. In N. M. Patrikalakis, editor, *Scientific Visualization of Physical Phenomena (Proceedings of CG International '91)*, pages 379–392. Springer-Verlag, 1991.

(Zhang and Webber 1993) Yuefeng Zhang and Robert E. Webber. Space diffusion: An improved parallel halftoning technique using space-filling curves. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 305–312, August 1993.